20.05.82.

COMAL DEFINITION - THE COMAL KERNAL

by

Borge R. Christensen

THE COMAL KERNAL

```
<comal program> ::=
            <block>

<block> ::=
          {<declaration statement> |
           <non declaration statement>}

<declaration statement> ::=
            <structured declaration statement> |
            <unstructured declaration statement>

<non declaration statement> ::=
            <structured statement> |
            <unstructured statement>

<structured declaration statement> ::=
            <procedure declaration> |
            <function declaration>

<unstructured declaration statement> ::=
            <dim statement> |
            <data statement>

<structured statement> ::=
            <repetitive statement> |
            <conditional statement>

<unstructured statement> ::=
            <simple statement> <eol> |
            <remark> <newline> |
            <label statement> <eol>

<eol> ::=
            [<remark>] <newline>

<remark> ::=
            //{<displayable character>}

<newline> ::=
            implementation dependent

<displayable character> ::=
            implementation dependent
```

```
<simple statement> ::=
            <stop statement> |
            <return statement> |
            <assignment statement> |
            <input statement> |
            <goto statement> |
            <restore statement> |
            <select statement> |
            <open statement> |
            <read statement> |
            <write statement> |
            <close statement> |
            <delete statement> |
            <print statement> |
            <zone statement> |
            <print using statement> |
            <procedure call statement>

<repetitive statement> ::=
            <while statement> |
            <repeat statement> |
            <for statement>

<conditional statement> ::=
            <if statement> |
            <case statement>

<while statement> ::=
            <short while statement> |
            <long while statement>

<short while statement> ::=
            WHILE <logical expression> DO <simple statement> <eol>

<long while statement> ::=
            WHILE <logical expression> DO <eol>
              <statement list>
            ENDWHILE <eol>


<statement list> ::=
           {<non declaration statement>}

<repeat statement> ::=
            REPEAT <eol>
              <statement list>
            UNTIL <logical expression> <eol>

<for statement> ::=
            <short for statement> |
            <long for statement>

<short for statement> ::=
            FOR <for range> [<step>] DO <simple statement> <eol>

<long for statement> ::=
            FOR <for range> [<step>] DO <eol>
              <statement list>
            NEXT <control variable> <eol>
```

```
<for range> ::=
            <control variable> := <initial value> TO <final value>

<step> ::=
            STEP <step value>

<control variable> ::=
            <numeric identifier>

<initial value> ::=
            <numeric expression>

<final value> ::=
            <numeric expression>

<step value> ::=
            <numeric expression>

<if statement> ::=
            <short if statement> |
            <long if statement>

<short if statement> ::=
            IF <logical expression> THEN <simple statement> <eol>

<long if statement> ::=
            IF <logical expression> THEN <eol>
              <statement list>
            {ELIF <logical expression> THEN <eol>
              <statement list>}
            [ELSE <eol>
              <statement list>]
             ENDIF <eol>

<logical expression> ::=
            <numeric expression>

<case statement> ::=
            CASE <case selector> OF <eol>
            WHEN <choice list> <eol>
              <statement list>
            {WHEN <choice list> <eol>
              <statement list>}
            [OTHERWISE <eol>
              <statement list>]
             ENDCASE <eol>

<case selector> ::=
            <expression>

<choice list> ::=
            <numeric expression> {,<numeric expression> } |
            <string expression> {,<string expression>}

<procedure declaration> ::=
            PROC <procedure identifier> <head appendix> <eol>
              <procedure block>
            ENDPROC <procedure identifier> <eol>
```

```
<function declaration> ::=
            FUNC <function identifier> <head appendix> <eol>
              <function block>
            ENDFUNC <function identifier> <eol>

<function block> ::=
            <procedure block>

<procedure block> ::=
            {<import statement>}
            {<unstructured declaration statement> |
            <non declaration statement>}

<head appendix> ::=
           [(<formal parameter list>)] [CLOSED]

<procedure identifier> ::=
            <identifier>

<function identifier> ::=
            <numeric identifier> |
            <string identifier>

<formal parameter list> ::=
            <formal parameter> {,<formal parameter>}

<formal parameter> ::=
            [REF] <variable identifier> |
            REF <variable identifier> <array indicator>

<import statement> ::=
            IMPORT <variable identifier> {,<variable identifier>} <eol>

<variable identifier> ::=
            <numeric identifier> |
            <string identifier>

<array indicator> ::=
            ({,})

<dim statement> ::=
            DIM <declaration> {,<declaration>} <eol>

<declaration> ::=
            <numeric declaration> |
            <string declaration>

<numeric declaration> ::=
            <numeric identifier> (<dimension part>)

<string declaration> ::=
            <string identifier> [(<dimension part>)] OF <length>

<dimension part> ::=
            <range> {,<range>}

<range> ::=
            [<lower bound>:] <upper bound>
```

```
<lower bound> ::=
            <numeric expression>

<upper bound> ::=
            <numeric expression>

<length> ::=
            <numeric expression>

<data statement> ::=
            DATA <value> {,<value>} <eol>

<value> ::=
            [<sign>] <integer> |
            [<sign>] <real number> |
            <string constant> |
            TRUE |
            FALSE

<sign> ::=
            + | -

<expression> ::=
            <string expression> |
            <numeric expression>

<numeric expression> ::=
            [<numeric expression> OR] <logical term>

<logical term> ::=
            [<logical term> AND] <logical factor>

<logical factor> ::=
            [NOT] <relation>

<relation> ::=
            <string relation> |
            <arithmetic relation>

<string relation> ::=
            <string expression> <relational string operator>
            <string expression>

<relational string operator> ::=
            IN | <relational operator>

<arithmetic relation> ::=
            <formula> [<relational operator> <formula>]

<relational operator> ::=
            < | <= | = | >= | > | <>

<formula> ::=
            [<sign>] <arithmetic expression>

<arithmetic expression> ::=
            [<arithmetic expression> <additive operator>] <term>
```

```
<additive operator> ::=
            + | -

<term> ::=
            [<term> <multiplicative operator>] <factor>

<multiplicative operator> ::=
            * |
            / |
            DIV |
            MOD

<factor> ::=
            <operand> [ <factor>]

<operand> ::=
            (<numeric expression>) |
            <constant> |
            <numeric variable> |
            <numeric function call>

<constant> ::=
            <integer> | <real number> |
            TRUE | FALSE

<real number> ::=
            <decimal number> [<exponent>]

<decimal number> ::=
            <integer> [. [<integer>]] |
            .<integer>

<exponent> ::=
            E [<sign>] <integer>

<integer> ::=
            <digit>{<digit>}

<numeric variable> ::=
            <numeric identifier> [(<subscript list>)]

<numeric identifier> ::=
            <integer identifier> |
            <real identifier>

<integer identifier> ::=
            <identifier>#

<real identifier> ::=
            <identifier>

<subscript list> ::=
            <subscript> {,<subscript>}

<subscript> ::=
            <numeric expression>

<numeric function call> ::=
            <numeric identifier> [(<actual parameter list>)]
```

```
<string expression> ::=
            <string operand> {+ <string operand>}

<string operand> ::=
            <string constant> |
            <string variable> |
            <string function call>

<string constant> ::=
            "{<displayable character>}"

<string variable> ::=
            <string identifier> [(<subscript list>)]
            [(<substring specifier>)]

<string identifier> ::=
            <identifier>$

<substring specifier> ::=
            <position> | <from>:<to>

<position> ::=
            <numeric expression>

<from> ::=
            <numeric expression>

<to> ::=
            <numeric expression>

<string function call> ::=
            <string identifier>[(<actual parameter list>)]
            [(<substring specifier>)]

<stop statement> ::=
            STOP

<return statement> ::=
            RETURN [<expression>]

<assignment statement> ::=
            <assignment> {;<assignment>}

<assignment> ::=
            <numeric assignment> |
            <string assignment>

<numeric assignment> ::=
            <numeric variable> := <numeric expression>

<string assignment> ::=
            <string variable> := <string expression>

<input statement> ::=
            INPUT [<string constant>:] <variable list> <print end> |
            INPUT <file designator>: <variable list>

<variable list> ::=
            <variable> {,<variable>}
```

```
<variable> ::=
            <numeric variable> |
            <string variable>

<file designator> ::=
            FILE <channel number> [,<record number>]

<channel number> ::=
            <numeric expression>

<record number> ::=
            <numeric expression>

<goto statement> ::=
            GOTO <label identifier>

<restore statement> ::=
            RESTORE [<label identifier>]

<select statement> ::=
            SELECT <type> <device specifier> [,<dev info>]

<type> ::=
            OUTPUT

<device specifier> ::=
            <string expression>

<open statement> ::=
            OPEN FILE <channel number>,<file name>
            [,<dev info>],<mode>

<dev info> ::=
            implementation dependent device information

<file name> ::=
            <string expression>

<mode> ::=
            READ |
            WRITE |
            APPEND |
            RANDOM <record length> [READONLY]

<record length> ::=
            <numeric expression>

<read statement> ::=
            READ <variable list> |
            READ <file designator>: <variable list>

<write statement> ::=
            WRITE <file designator>: <variable list>

<delete statement> ::=
            DELETE <file name> [,<dev info>]

<close statement> ::=
            CLOSE [FILE <channel number>]
```

```
<print statement> ::=
            PRINT [<output list>] |
            PRINT <file designator>: [<output list>]

<output list> ::=
            <print list> [<print end>]

<print list> ::=
            <print element> {<print separator> <print element>}

<print element> ::=
            <expression> |
            <tab function>

<print end> ::=
            <print separator>

<print separator> ::=
            , | ;

<tab function> ::=
            TAB(<numeric expression>)

<zone statement> ::=
            ZONE <numeric expression>

<print using statement> ::=
            PRINT USING <format info>: <using list> [<print end>] |
            PRINT <file designator>: USING <format info>:
            <using list> [<print end>]

<using list> ::=
            <using element> {,<using element>}

<using element> ::=
            <numeric expression>

<format info> ::=
            <string expression>

<procedure call statement> ::=
            [EXEC] <procedure identifier> [(<actual parameter list>)]

<actual parameter list> ::=
            <actual parameter> {,<actual parameter>}

<actual parameter> ::=
            <expression>

<label statement> ::=
            <label identifier>:

<label identifier> ::=
            <identifier>

<identifier> ::=
            <letter> {<letter> | <digit> | ' }
```

```
<letter> ::=
            implementation dependent

<digit> ::=
            0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

APPENDIX A

STANDARD BUILT IN FUNCTIONS

```
ABS(<numeric expression>) |
COS(<numeric expression>) |
SIN(<numeric expression>) |
TAN(<numeric expression>) |
ATN(<numeric expression>) |
LOG(<numeric expression>) |
EXP(<numeric expression>) |
SQR(<numeric expression>) |
INT(<numeric expression>) |
EOF(<numeric expression>) |
SGN(<numeric expression>) |
RND[(<numeric expression>)] |
RND(<numeric expression>,<numeric expression>) |
LEN(<string expression>) |
ORD(<string expression>) |
EOD |
ZONE |
VAL(<string expression>) |
STR$(<numeric expression>) |
CHR$(<numeric expression>)
```

COMAL COMMANDS IN AN INTERACTIVE ENVIRONMENT.

```
<command> ::=
            <new command> |
            <run command> |
            <continue command> |
            <list command> |
            <automatic line numbering command> |
            <renumber line command> |
            <delete line command> |
            <store program command> |
            <retrieve program command> |
            <workspace size request>

<new command> ::=
            NEW <eol>

<run command> ::=
            RUN <eol>

<continue command> ::=
            CON <eol>

<list command> ::=
            LIST [<line number range>] <eol>

<line number range> ::=
            <line number> [-[<line number>]] |
            -<line number>

<line number> ::=
            implementation dependent positive integer

<automatic line numbering command> ::=
            AUTO [<line numbering specifier>] <eol>

<line numbering specifier> ::=
            [<start>] [,<increment>]

<start> ::=
            <line number>

<increment> ::=
            <integer>

<renumber line command> ::=
            RENUM [<line numbering specifier>] <eol>

<delete line command> ::=
            DEL <line number range> <eol>

<store program command> ::=
            SAVE <file name> [,<dev info>] <eol> |
            LIST [<line number range>] <file name> [,<dev info>] <eol>
```

```
<retrieve program command> ::=
            LOAD <file name> [,<dev info>] <eol> |
            ENTER <file name> [,<dev info>] <eol>

<workspace size request> ::=
            SIZE <eol>
```

SUGGESTED EXTENSIONS TO COMAL KERNAL.

Extensions to:

```
<numeric assignment> ::=
            <numeric variable> :+ <logical expression> |
            <numeric variable> :- <logical expression>

<string assignment> ::=
            <string variable> :+ <string expression>

<stop statement> ::=
            STOP <string expression>

<simple statement> :=
            <chain statement> |
            <null statement> |
            <exit statement> |
            <cursor control statement> |
            <clear screen statement> |
            <form feed statement>

<chain statement> ::=
            CHAIN <file name> [,<dev info>]

<null statement> ::=
            NULL

<exit statement> ::=
            EXIT

<cursor control statement> ::=
            CURSOR <row>,<column>

<row> ::=
            <logical expression>

<column> ::=
            <logical expression>

<unstructured declaration statement> ::=
            <local statement> |
            <use statement> |
            <discard statement>

<local statement> ::=
            LOCAL <local declaration> {,<local declaration>} <eol>
```

SUGGESTED EXTENSIONS TO COMAL KERNAL.

Extensions to:

```
<numeric assignment> ::=
            <numeric variable> :+ <logical expression> |
            <numeric variable> :- <logical expression>

<string assignment> ::=
            <string variable> :+ <string expression>

<stop statement> ::=
            STOP <string expression>

<simple statement> :=
            <chain statement> |
            <null statement> |
            <exit statement> |
            <cursor control statement> |
            <clear screen statement> |
            <form feed statement>

<chain statement> ::=
            CHAIN <file name> [,<dev info>]

<null statement> ::=
            NULL

<exit statement> ::=
            EXIT

<cursor control statement> ::=
            CURSOR <row>,<column>

<row> ::=
            <logical expression>

<column> ::=
            <logical expression>

<unstructured declaration statement> ::=
            <local statement> |
            <use statement> |
            <discard statement>

<local statement> ::=
            LOCAL <local declaration> {,<local declaration>} <eol>

<local declaration> ::=
            <declaration> |
            <numeric identifier>

<use statement> ::=
            USE <package identifier> <eol>
```

```
<discard statement> ::=
            DISCARD <package identifier> <eol>

<package identifier> ::=
            <identifier>

<repetitive statement> ::=
            <loop statement> |
            <short repeat statement>

<loop statement> ::=
            LOOP <eol>
              <statement list>}
            ENDLOOP <eol>

<short repeat statement> ::=
            REPEAT <simple statement>
            UNTIL <logical expression> <eol>

<short if statement> ::=
            IF <logical expression> THEN <simple statement>
            ELSE <simple statement> <eol>

<procedure declaration> ::=
            PROC <procedure identifier> <external head appendix> <eol>

<function declaration> ::=
            FUNC <function identifier> <external head appendix> <eol>

<external head appendix> ::=
            [(<formal parameter list>)] <external specifier>

<external specifier> ::=
            EXTERNAL <string constant> [,<dev info>]

<clear screen statement> ::=
            PAGE

<form feed statement> ::=
            PAGE
```

Tonder, Denmark, 20 May 1982 Borge R. Christensen